An Agent-Based Framework for Collaborative Data Mining Optimization

Xiong Liu, Kaizhi Tang
Intelligent Automation, Inc.
Rockville, MD 20855, USA
xliu09@gmail.com

John R. Buhrman, Huaining Cheng
Air Force Research Laboratory
OH 45433, USA

ABSTRACT

Data mining meta-optimization aims to find an optimal data mining model which has the best performance (e.g., highest prediction accuracy) for a specific dataset. The optimization process usually involves evaluating a series of configurations of parameter values for many algorithms, which can be very time-consuming. We propose an agent-based framework to power the meta-optimization through collaboration of computing resources. This framework can evaluate the parameter settings for a list of algorithms in parallel via a multi-agent system and therefore can reduce computational time. We have applied the framework to the construction of prediction models for human biomechanics data. The results show that the framework can significantly improve the accuracy of data mining models and the efficiency of data mining meta-optimization.


1. INTRODUCTION

Data mining is the process of automatically searching large volumes of data for hidden patterns using algorithms such as classification, clustering and association rule mining. A key objective of data mining is to build data mining models [1]. For example, biomechanics is an emerging area where model building can provide significant advantage. As more data are collected from biomechanics experiments, simulations and domain rules, it is becoming an important area to apply data mining technology to biomechanics data for human performance and injury prevention. With the extracted knowledge built by prediction or classification models, quick response to "what-if" questions will be provided to tell either the strength of human response or injury risk under different motion scenarios [2, 3].

Given a specific dataset and question in mind, there can be a number of data mining algorithms suitable for model building, each of which has its own parameter set. To build a high-quality model, it is necessary to select the right algorithm with a right set of parameter values. Algorithm and parameter selection can be treated as a meta-optimization problem, i.e., finding an optimal model which has the best performance (e.g., highest prediction accuracy) for a specific dataset. The data mining meta-optimization problem can take place at two levels: parameter level and algorithm level. The parameter-level optimization refers to finding the parameter settings that will result in optimal performance for a given algorithm. The algorithm-level optimization refers to selecting the algorithm with best performance from a list of applicable algorithms, each of which is considered for parameter optimization.

The data mining meta-optimization process usually involves evaluating a series of configurations of parameter values for multiple algorithms. This can be very time-consuming depending on the size of the problem. Since each configuration can be evaluated independently, the data mining meta-optimization problem can be solved using distributed computing. In this paper, we introduce an agent-based framework for collaborative data mining meta-optimization. The framework is designed to support flexible distributed computing using multi-agent systems (MASs) and collaboration of computing resources. Agents are software components that embody the characteristics of: (1) goal orientation, with task focus and autonomy, (2) social-ability, engaged in communications and coordination with other agents, and (3) adaptive [4]. MAS is a system composed of several agents, capable of reaching goals that are difficult to achieve by an individual agent [5, 6]. The computation is asynchronous in MASs. The agents interact with each other according
to a conversational abstraction; agents send messages to each other when they need to communicate in order to execute a task. This framework also enables collaboration of computing resources and data resources by allowing users to register their computers with collaborative agents. Due to the flexibility and scalability features of agents and MASs, our system is capable of handling large-scale data mining problems (e.g., evaluating a large number of algorithms for a dataset) by allocating tasks to the agents installed on individual computers.

The rest of the paper is organized as follows. In Section 2, we provide background on data mining meta-optimization. In Section 3, we describe the agent-based framework. In Section 4, we discuss the analysis of biodynamic data using data mining meta-optimization and report performance measurement of the agent-based framework. Finally, we conclude the paper in Section 5.

2. DATA MINING META-OPTIMIZATION

Most data mining algorithms require the setting of input parameters. For example, the parameters in a support vector machine (SVM) include penalty parameters and the kernel function parameters. These parameters usually have significant influences on the performance of the algorithm.

The problem of finding the best value for each parameter of a chosen algorithm can be viewed as a state-space search problem, which can be characterized by five components: state space $S$, objective function $F(S)$, operator $O$, initial state $S_0$, and termination condition $T$ [7]. A state space $S$ represents the set of states explored by the search. Here, a state is a parameter setting. The goal of the search is to find the optimal state satisfying the constraints of the value range. The objective function $F(S)$ is a function mapping a state to the real numbers. It can be the prediction error (based on k-fold cross-validation) or computational cost of a state. An operator $O$ is a function mapping one state into another or transforming one parameter setting to another. It is dependent on the search strategy adopted. The initial state $S_0$ is the starting point for the search. Here, it is the default parameter setting of the algorithm. Termination condition $T$ is the criterion for stopping the search, by which a state is accepted as the best state if further search in the space does not return a better state.

Theoretically, a state space $S$ should cover all possible combinations of parameter values. When the number of combinations is relatively small, an exhaustive search can be applied. However, when the number of combinations is large, the search work can only be performed on a subset of the entire space of parameters. The heuristic techniques based on Genetic Algorithm (GA) have been reported to be effective in optimizing SVM [8] and neural networks [9]. They presumably can also be applied to other algorithms.

With the parameter optimization embedded, the process of algorithm-level optimization iterates within a list of feasible algorithms and finds one or multiple algorithms with highest prediction accuracy. Since there are multiple algorithms, each with multiple parameter settings, the optimization process involves expensive computation load (for both exhaustive search and heuristic search). This calls for scalable solution with distribution computation.

3. AGENT-BASED FRAMEWORK FOR DATA MINING META-OPTIMIZATION

We have developed an agent-based framework to power the meta-optimization through collaboration of computing resources. This framework can evaluate the parameter settings for a number of algorithms in parallel via a multi-agent system and therefore can reduce computational time.

Several efforts have been devoted to data mining through agent-based systems. These systems generally fall into two categories: Centralized systems and Peer-to-Peer (P2P) oriented systems [10]. Centralized systems usually include a facilitator agent which interacts with the mining agents. P2P based architectures exploits P2P concepts to improve system capacity. Here, we present a simple yet extensible centralized architecture for meta-optimization.

3.1. System Architecture

Figure 1 shows the architecture of our framework, where user applications, e.g., Java programs running in a Java...
Virtual Machine (JVM), are flexibly coupled with a MAS. The MAS consists of a manager agent and several worker agents. The manager agent is responsible for handling and allocating tasks, i.e., model building and cross validation. The work agents are computational engines that perform specific tasks. There are several levels of messaging between the user application, the manager agent and the worker agents. When the MAS is started, the manager agent will send a message to all worker agents to query their processing capabilities (e.g., computation). The manager will then create an agent monitoring activity for each worker agent. When the user application (i.e., meta-optimization) is started, it will send the list of algorithms and the dataset to the manager agent. Once the manager agent receives this message, it will parse the application into a list of tasks, where each task \( T_i \) consists of a model (to be built) and a dataset. The tasks are stored in a dynamic data structure called Task Queue. This queue is a common object that can be accessed by the worker agents according to predefined protocols (see Section 3.2). Once the tasks are allocated, the manager agent will wait for the results from individual worker agents. When the manager agent obtains the results from all worker agents, it will deliver the combined results to the user application. Since the tasks are performed concurrently on the worker agents, the total response time is expected to be significantly reduced when compared with sequential processing.

In addition, this architecture supports collaboration among users. Users can register their computers with work agents to share their computing power. Also, users can contribute algorithms and datasets using the algorithm agent and the data agent. The algorithms and datasets transmitted to the manager agent could be discovered by the user application so that users can select the algorithms and/or datasets that are of their interest.

### 3.2 Design of Agent Protocols

The distribution of data mining tasks requires the design of agent interaction protocols. In theory, the interactions between agents can be either cooperative or selfish. That is, the agents can share a common goal (e.g., an ant colony), or they can pursue their own interests (e.g., the free market economy). To realize flexible distributed computation, our design of agent interactions adopts both cooperation and competition mechanisms. In the cooperation mechanism, the worker agents share the same goal of performing tasks stored in the task queue. A worker agent will first check the availability of tasks. If the queue is not null, the worker agent will remove the top task from the queue and perform it. In the competition mechanism, the worker agents compete with each other to access the tasks according to the rule of “first in first out” (FIFO). That is, a worker agent is allowed to access the queue and perform new tasks as long as it finishes a previous task.

![Figure 2. Task Allocation between Worker Agents](image)

Figure 2 depicts an example of task allocation for the model building and cross-validation scenario. In this example, there are three tasks (cross validation of three models) in the queue. And there are only two worker agents: “Worker 1” and “Worker 2”. When the task queue is initialized, the manager agent will send a notifying message to the workers. Both workers will obtain a task in the first round. In this case, Worker 1 obtains task \( T_1 \) (cross validation of a regression model), and Worker 2 obtains task \( T_2 \) (cross validation of a KStar model). Assuming Worker 1 has more computing resources and completes its task earlier than Worker 2, it will access the task queue for the second time and obtain task \( T_3 \) (cross validation of a tree model). That is, Worker 1 wins the competition by performing more tasks.

To estimate the response time, we denote the time required to finish task \( T_i \) (\( i=1\ldots M \)) as \( t_i \), and the number of worker agents as \( j (j=1\ldots N) \). We only consider the short-of-resource situation where the number of tasks exceeds the number of worker agents, or \( M > N \). For example in Figure 2, there are 3 tasks and only 2 worker agents. Assuming \( T_1 \) takes the shortest time and \( T_3 \) takes the longest time (\( t_1 < t_2 < t_3 \)), the time taken by Worker 1 will be \( (t_1 + t_3) \), and the time taken by Worker 2 will be \( t_2 \). Since \( (t_1 + t_3) \) is longer than \( t_2 \), the total waiting time for the manager agent will be \( (t_1 + t_3) \).

### 3.3 Implementation

We have developed a meta-optimization tool using IAI’s Cybele\textsuperscript{TM} agent platform [11] and RapidMiner (formerly YALE: Yet Another Learning Environment) [12]. Cybele is agent infrastructure for developing Java-based agent applications and provides many useful application programming interfaces (APIs). It allows us to focus on the design of our application without having to worry about lower-level communication details. RapidMiner is an open-source Java-based environment for complex data mining processes. It provides many utilities (i.e., operators) such as data transformation, model
construction and model validation. Hence, our optimization tool is a synergy of Cybele and RapidMiner capabilities.

The Cybele agent system consists of multiple worker agents and a manager agent. These agents are created using the Cybele APIs, each with a predefined role. Worker agents are computational units for specific data mining tasks (e.g., cross-validation). They can run in different Java Virtual Machines (JVMs), either in the same machine or separate machines in the same network. The manager agent is a moderator of computational resources which manages a group of worker agents. The manager agent receives the parameter settings and dataset from RapidMiner and then allocates the cross validation tasks (one task for one parameter setting) to worker agents.

To enable parameter optimization, we developed an operator called ParameterOptimization within RapidMiner. This operator can generate the cross-validation tasks for the different parameter settings of a learning algorithm. The tasks are then distributed over the Cybele agent system for parallel computation. For example, in Figure 3, there are three values for each of the three parameters (M, N, and S) of the W-REPTree algorithm. The ParameterOptimization operator will parallel cross-validate the $3^3 = 27$ parameter settings via the agent framework and return an optimal set of parameter values (e.g., $M = 1$, $N = 3$, and $S = 1$).

XML fragment:

```xml
<list key="parameters">
  <parameter key="W-REPTree.M" value="1.0,2.0,3.0"/>
  <parameter key="W-REPTree.N" value="2.0,3.0,4.0"/>
  <parameter key="W-REPTree.S" value="1.0,2.0,3.0"/>
</list>
```

Figure 3. (A) XML Code for Defining Parameters; (B) GUI for Defining Parameters

To support parallel processing of the algorithm-level optimization, we designed an operator called AlgorithmOptimization within RapidMiner. This operator integrates parameter-level optimization and algorithm-level optimization into one unit. It can read the parameter settings of all algorithms and can distribute the evaluation of those settings via the agent framework. This operator also has a table that maintains the evaluation results. For each algorithm, the table stores the best set of parameter values discovered so far.

Figure 4 shows the RapidMiner interface of agent-based data mining meta-optimization. Note that the algorithm-level optimization operator contains several parameter-level optimization operators, each for a different algorithm.

4. EXPERIMENTS

To show the benefit of utilizing an agent framework for data mining meta-optimization, we have applied our agent framework to the construction of prediction models for biomechanics data. We configured the framework to run on personal computers connected in a local area network with one agent in one personal computer (Intel Core 2 CPU, 2GB of RAM).

4.1 Dataset and Algorithms

The Vertical Deceleration Tower (VDT) is used by the Air Force to provide an impact impulse to test subjects
We selected five applicable learning algorithms from the RapidMiner package. These algorithms can process both nominal and numeric attributes and can predict numeric class labels. Table 1 shows the parameters for the selected algorithms.

### Table 1. Selected Learning Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Sample Parameters</th>
<th>Sample Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian-Processes</td>
<td>L – level of Gaussian noise</td>
<td>L = 1, 2, 3, 4, 5, 10</td>
</tr>
<tr>
<td></td>
<td>N – whether to 0=normalize, 1=standardized, 2=neither</td>
<td>N = 0, 1, 2</td>
</tr>
<tr>
<td>IBK</td>
<td>K – number of nearest neighbors</td>
<td>K = 1, 2, 3, 4, 5</td>
</tr>
<tr>
<td></td>
<td>W – maximum number of training instances maintained</td>
<td>W = 0, 1, 2, 3</td>
</tr>
<tr>
<td></td>
<td>F – weight neighbors by 1-their distance</td>
<td>F = true, false</td>
</tr>
<tr>
<td></td>
<td>E – minimize mean squared error</td>
<td>E = true, false</td>
</tr>
<tr>
<td>KStar</td>
<td>B – manual blend setting (percentage)</td>
<td>B = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60</td>
</tr>
<tr>
<td></td>
<td>M – missing value treatment mode; valid options are ‘a’=average, ‘d’=delete, ‘m’=maxdiff, ‘n’=normal</td>
<td>M = ‘a’, ‘d’, ‘m’, ‘n’</td>
</tr>
<tr>
<td>M5P</td>
<td>M – minimum number of instances per leaf</td>
<td>M = 1, 2, 3, 4, 5, 10</td>
</tr>
<tr>
<td></td>
<td>N – use unpruned tree/rules</td>
<td>N = true, false</td>
</tr>
<tr>
<td></td>
<td>U – use unsmoothed predictions</td>
<td>U = true, false</td>
</tr>
<tr>
<td></td>
<td>R – build regression tree rather than a model tree</td>
<td>R = true, false</td>
</tr>
<tr>
<td>REPTree</td>
<td>M – minimum number of instances per leaf</td>
<td>M = 1, 2, 3</td>
</tr>
<tr>
<td></td>
<td>N – number of folds</td>
<td>N = 2, 3, 4</td>
</tr>
<tr>
<td></td>
<td>S – seed for random data shuffling</td>
<td>S = 1, 2, 3</td>
</tr>
<tr>
<td></td>
<td>P – no pruning</td>
<td>P = true, false</td>
</tr>
</tbody>
</table>

The first column is the name of algorithms; the second column is the list of the parameters; the third column is the possible values of the parameters. “GaussianProcesses” is an algorithm that implements Gaussian Processes for regression without hyperparameter-tuning [13]. Its parameters include the level of Gaussian noise (real; default: 1) and the ways to process data (real; default: 0=normalize). “IBK” is a K-nearest neighbor classifier [14]. Its parameters include the number of nearest neighbors (real; default: 1), the window size or the maximum number of training instances maintained (real; default: 0=no window), whether to weight neighbors by 1-their distance (Boolean; default: false), and whether to minimize mean squared error rather than mean absolute error for numeric prediction (Boolean; default: false). “KStar” is an instance-based classifier where the class of a test instance is based upon the class of those training instances similar to it [15]. “KStar” differs from other instance-based learners in that it uses an entropy-based distance function. Its parameters include the blending parameter that specifies the effective number of instances used for entropy calculation (real; default: 20%) and the missing value treatment mode (string; default: ‘a’=average). “M5P” is an algorithm that implements base routines for generating M5 Model trees and rules. The original algorithm M5 was invented by R. Quinlan [16] and Yong Wang made improvements [17]. M5P’s parameters include the minimum number of instances per leaf (real; default: 4), whether to use unpruned tree/rules (Boolean; default: false), whether to use unsmoothed predictions (Boolean; default: false), and whether to build regress tree rather than a model tree (Boolean; default: false). “REPTree” is a fast decision tree learner. It is a mixture of decision tree and linear regression, where each leaf node corresponds to a linear regression algorithm [18]. Its parameters include the minimum number of instances per leaf (real; default: 2), the number of folds for reduced error pruning (real; default: 3), the seed for random data shuffling (real; default: 1) and whether to use unpruned tree (Boolean; default: false).

#### 4.2 Prediction Performance

We created a RapidMiner workflow according to the parameter configuration shown in Table 1. Together there are 248 parameter settings for the five algorithms, including the default parameter settings. The prediction results based on 10-fold cross-validations of the parameter settings are shown in Table 2.

### Table 2. Prediction Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No Optimization</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Root means squared error</td>
<td>Relative absolute error</td>
</tr>
<tr>
<td>Gaussian-Process</td>
<td>443.686</td>
<td>0.695</td>
</tr>
<tr>
<td>IBK</td>
<td>270.488</td>
<td>0.270</td>
</tr>
<tr>
<td>KStar</td>
<td>284.238</td>
<td>0.418</td>
</tr>
<tr>
<td>M5P</td>
<td>273.295</td>
<td>0.314</td>
</tr>
<tr>
<td>REPTree</td>
<td>343.693</td>
<td>0.265</td>
</tr>
</tbody>
</table>

Without parameter optimization (using the default parameter settings), REPTree has the lowest relative absolute error (26.5%). With parameter optimization, REPTree still has the lowest prediction error but the error
has decreased to (17.1%). Other algorithms have also seen a decrease in prediction error with parameter optimization. The results indicate that meta-optimization can affect the performance of an algorithm and therefore the model building and selection.

4.3 Time Analysis for Agent Configurations

We evaluated the 248 parameter settings using a different number of worker agents. The execution time for the cross-validations of the 248 parameter settings is stored in a time log file on the manager agent.

![Figure 5. The Functionality between the Execution Time and the Number of Agents](image)

Figure 5 shows the execution time as a function of the number of worker agents. The time is averaged based on 10 runs. As can be seen, the execution time is 1481.8 seconds when using just one worker agent. The time drops significantly to 596.02 seconds when using three worker agents. When using more than five worker agents, the time continues to drop but less significantly. When using more than nine worker agents, the time becomes stable around 290 seconds. The result shows that using more worker agents can significantly reduce the execution time of cross-validation tasks when compared with using one worker agent. However, for the problem size of 248 tasks, the drop of time becomes less effective when using more than five worker agents.

4.4 Time Analysis for Different Number of Tasks

To measure the functionality between the execution time and the number of tasks, we randomly sampled a fixed number of tasks (e.g., 50) from the task population and cross-validated them under different agent configurations.

![Figure 6. The Functionality between the Execution Time and the Number of Tasks](image)

Figure 6 shows the execution time (averaged based on 10 runs) as a function of the number of tasks. It can be seen that there is a linear increase in time as the number of tasks increases. Also, the slope of time increase becomes smaller as the number of agents increases. This result shows that using more agents can improve the scalability of the agent framework. However, when using more than 5 agents, the gain in scalability is not significant.

5. CONCLUSION

Data mining meta-optimization is a critical step in building high-quality data mining models. By exploring the parameter settings for applicable algorithms, it enables the data analyst to find optimal data mining models for a specific dataset. We have applied parameter-level optimization to five algorithms in modeling building for a human VDT dataset. The results show that meta-optimization can improve the accuracy of data mining algorithms.

The data mining meta-optimization problem for algorithm and parameter selection involves expensive computation load. Both of them call for scalable solution with distributed computation. We have developed an agent-based framework to power the meta-optimization through collocation and conducted experiments using the framework. We measured the total execution time of 248 cross-validation tasks (averaged based on 10 runs) associated with meta-optimization under different agent configurations. When using 5 agents with one agent in one personal computer (Intel Core 2 CPU, 2GB of RAM), the total execution time is only about 1/6 of the centralized approach (1 agent) for 248 tasks, which demonstrates the great benefit of utilizing an agent framework for collaborative data mining meta-optimization.
Future work will include the design of interfaces for collaboration, visualization methods for collaborative data mining, web infrastructure for collaboration (e.g., sharing of data mining results), and more different applications of the proposed agent-based framework.

ACKNOWLEDGEMENT

The authors wish to thank Air Force Research Laboratory, Human Effectiveness Directorate for their support under research contract No. FA8650-06-M-6709 and FA8650-07-C-6761.

REFERENCES


